On Middleware

History repeats itself in middleware debates

October 1998/By Max Dolgicer

I had lunch with a client recently. After a pleasant conversation covering everything from the weather (a permanent table discussion) to Clinton's troubles, we got down to business -- a discussion of which middle-tier middleware would be best for a specific three-tier showcase application his company wanted to design. He wanted to use Java (though this was still under discussion) as a language for server-side development, and was looking at browser and regular client access to this middle tier. He reminded me that he is a strong believer in applications that consist of objects and components.

On the plus side, his middleware did not have to support high transaction volume. Most of his company's transactional activities, such as database updates, would remain on the mainframe, where everything is simpler. Last but not least, he wanted to build a real infrastructure along the lines of the application. Thus, from a middleware perspective, many pieces could be reused for future applications.

However, his next statement gave me food for thought for weeks to come. I was expecting to continue discussing subjects like whether Java is mature enough as a server-side language and whether the project should use a commercial application server, such as those from NetDynamics or Netscape, or a plain Java/Common Object Request Broker (Corba) solution. Instead, he bent over the table and said, "There will be no product decision at this time." He then listed a few reasons for the decision: technology turmoil and overall confusion in the middleware market, and some licensing issues resulting from a planned merger of his firm and another company. For the next few months, he said, his firm would be focusing on developing applications using less-glamorous but more pragmatic technologies where a clear benefit to business could be easily quantified. For example, he said, instead of trying to choose middleware for the middle tier, he was now more interested in investing in technologies such as message brokers and workflow engines.

While I was at first surprised by this turn of events, the subway ride back to the office gave me ample opportunity to ponder his decision. I judged that this client is not alone in his thinking. He and others like him are concluding that "wait and see" is the best decision in I/S today. Fierce start-up, merger and acquisition activity in the software industry fuels doubts about which vendors will survive the long haul. Mergermania among large and small corporations makes any serious technology decision very risky, if not impossible. And the middleware arena is one of the most hotly contested software sectors today. The selection of a middleware platform (yes, middleware is a platform) is vital to a company, even more important than the choice of a middle-tier OS.

Despite all this, some smart people recommend that I/S executives put a stake in the middleware ground and move forward. In principle I agree with this strategy, but I will also concede that history usually repeats itself and cannot be ignored. It was much easier to make technology decisions in past years. Let's take a look at the history of middleware debates for some guidance.

Apple pie, RPC and MOM

Just a few years ago, the middleware debate was framed around the very simple question of Remote Procedure Call (RPC) vs. message-based products. Most middleware products solved a couple of very basic problems, such as program-to-program communication. And many middleware products furnished developers with higher level APIs that completely alleviated the need for cumbersome network protocols and APIs like TCP sockets or SNA LU 6.2.

RPCs, for example, allow functions to be called synchronously without regard to whether these functions are local or remote. RPC-based middleware spares developers the chore of writing custom program-to-program communication and marshaling code. Both of these are generated from Interface Definition Language (IDL) in the form of client and server stubs.

In turn, messaging-based products expand the otherwise synchronous and connection-oriented style of program-to-program communication to different asynchronous approaches, such as message queuing and publish/subscribe. In general, though, most messaging products do not lift the burden of reconciling data representation between machine types from the developer's shoulders. While messaging products can provide more flexibility in the way application programs communicate with each other, this middleware type cannot generate any communication code. On the contrary, programmers have to write to messaging APIs provided by commercial messaging products to enable program-to-program communication.

Looking back at the mix of products and firms that were part of that "great debate," a couple of questions come to mind: a) How many of these products and firms are still around, and b) How important is the RPC vs. messaging debate today? The answer to the first question is simple -- not many such firms are still around (substantial credit goes to the survivors). The answer to the second question is more complex.

By now, we have already decided that a formidable middleware product should be "best of breed." It should support synchronous, connection-based communication promoted by RPC-based products as well as asynchronous, connectionless communication supported by message-oriented middleware (MOM). In fact, there are products sold today that incorporate these two distinct characteristics. And more are on the way. The Object Management Group (OMG) is incorporating capabilities found previously only in messaging products into its Corba 3 specifications. In turn, capabilities found today in Microsoft's MTS, Component Object Model (COM) and MSMQ technologies will be incorporated into Microsoft's COM+ specifications, allowing any methods to be invoked synchronously or asynchronously without compromising functionality. So the correct answer to the second question is that credible middleware products must support capabilities previously available in either RPC or messaging products.

The RPC vs. messaging debate ended with IBM's MQSeries MOM middleware declared the winner and the Distributed Computing Environment (DCE) pronounced dead. (IBM, equally committed to both technologies, could not win twice.) The debate then moved on to object middleware, where Corba and a number of compliant products clashed directly with Microsoft's COM.

Interestingly enough, the COM vs. Corba debate has overshadowed the continued evolution of MOM in its own world. IBM's revenue from MQSeries continues to grow aggressively. The Corba vs. COM debate has also had no affect on sales of TP monitors. This reinforces the fact that when it comes to production applications, stability, scalability and reliability are more important than any "religious war," including COM vs. Corba and object vs. procedural middleware.

Nevertheless, giving developers the ability to define standard, high-level interfaces to objects in a programming-language-independent fashion is very appealing. Similarly, the concept of a standard infrastructure that spells out how objects communicate is also appealing. These factors, combined with the fact that many of us are in continuous pursuit of new challenges, gave stage to the COM vs. Corba debate. While Corba and COM are different object models, both pursue two important goals -- providing standard interfaces to define, traverse and query object interfaces, and providing a solid infrastructure for distributed objects.

The COM vs. Corba debate quickly became far more zealous and emotional than the previous RPC vs. messaging contest. Because any large-scale application requires I/S management to select a middleware technology, the stakes are always high. And when one side is Microsoft, the stakes are even higher. Think about it: I/S managers must not only decide whether object or more traditional middleware will be used for a project, but also whether Corba or COM will be the backbone of production applications. Often, this decision is made when an operating system is selected. Since I don't know of many companies using Unix or Network Computers (NCs) on the client side, the operating system battle generally takes place in the middle tier, where NT and Unix usually clash.

In theory, middleware is indifferent to operating systems. One of the first reasons middleware was created was to create a level playing field for application developers where services (such as security, naming and transaction) and their respective APIs are available on many operating systems. Even Microsoft's COM is available on different Unix flavors and mainframe systems. In reality, COM, Corba and operating system decisions are tightly coupled. Because Microsoft believes that middleware functionality belongs to the operating system, it plans to bundle them.

As time passes by, we find more middleware integrated with Microsoft operating systems.

Today, because of its scalability limitations, NT does not pose serious competition to Unix for large-scale, mission-critical applications. Thus, Corba is the logical middleware choice for proponents of object middleware. An NT decision makes Corba harder to justify. The first problem is the additional cost. By bundling COM, MTS and MSMQ with an OS, Microsoft redefined middleware pricing and total-cost-of-ownership models. Corba requires additional license fees. Second, COM is part of, and very well integrated with, popular development tools, including Visual Basic and PowerBuilder. Choose Corba and additional development dollars will have to be dedicated to solve the problem of accessing Corba objects from Visual Basic clients.

Does this mean that if your middle-tier operating system is NT, COM is your middleware of choice? Not quite yet. There are a number of reasons why Corba should still be considered. These include avoiding operating system lock-in and accommodating the existence of many Unix applications that must integrate with newly developed NT applications. These reasons should be enough to justify the additional cost of a Corba license. Remember, history repeats itself and it appears that the COM vs. Corba debate is heading the way of the RPC vs. messaging contest. It is inevitable that most large enterprises will end up with both COM and Corba.

One-stop shopping?

No industry is more dynamic than information technology. There is never a dull moment. In the best-case scenarios, the questions remain the same as the answers change. In reality, both questions and answers constantly change. The arrival of Java combined with the ambiguity of the Web produced a number of new, yet-to-be-answered questions: Is Java the answer to all middleware decisions? and Is Java middleware? No one can argue that the original existence of middleware was intimately linked to the problem of portability. Further, Java's popularity combined with the ambiguity of the Web fostered the new analyst term "Web-centric middleware." Products like NetDynamics 4.0 (acquired by Sun), Netscape Application Server (formerly Kiva), Tengah from WebLogic, and Oracle's just-announced Application Server 4.0 are examples of

Web-centric middleware.

Generally, these products provide some sort of one-stop shopping -- they offer development, deployment and management environments. Although the marketing efforts of these companies have so far focused on Web applications (HTML and Java support for the client, and Java support for the server), there is no technical reason why such products cannot participate in the battle for the middle tier of non-Web applications. In fact, Microsoft has already positioned MTS as a universal middleware technology capable of providing a home for middle-tier components serving Web and non-Web applications.

As the sophistication of these products increases, so does the appetite of the suppliers for "sphere of influence." To assume that the marketing focus of these companies will be Web-only applications is simply naive. By the same token, it would be wishful thinking to assume that well-established middleware vendors, such as BEA Systems, would not attempt to extend the reach of middleware products like Tuxedo and M3 to pure Java/Web applications.

The question is this: Can a single vendor’s technology successfully convince developers that its products cover all aspects of development, deployment and management in applications spanning universal clients, servers and legacy access? Look for the answers in a future column.

---

illustration by mike brennan
